KOCL:
Kernel-level Power Estimation for Arbitrary FPGA-SoC-accelerated OpenCL Applications

James Davis, Josh Levine, Ed Stott, Eddie Hung, Peter Cheung and George Constantinides
Imperial College London

james.davis@imperial.ac.uk
Executive Summary

• **KAPow for OpenCL**
  – ‘K’ounting Activity for Power Estimation
• Hardware/software framework providing **kernel-level power estimates** for **OpenCL applications** running on **Altera FPGAs**
• Trains, adapts online with real workload
• Up to ±5mW accuracy
• Fully automated
• Minimalist API
• Open source
  – [https://github.com/PRiME-project/KOCL](https://github.com/PRiME-project/KOCL)
KOCL: Power Self-Awareness for Arbitrary FPGA-SoC-Accelerated OpenCL Applications

James J. Egbe
Imperial College London
Joshua M. Levyke
Udit
Edward A. Saff
Imperial College London
Eddy Hoang

Peter Y. K. Cheung and
George A. Constantinides
Imperial College London

KOCL is a new approach to power self-awareness in OpenCL. It enables developers to monitor and manage the power consumption of OpenCL applications in real-time. KOCL uses a novel approach to calculate the power consumption of OpenCL kernels and provides feedback to the developer about the power usage. This allows developers to optimize their applications for power efficiency and ensures that the power consumption remains within acceptable limits. KOCL is designed to work seamlessly with existing OpenCL frameworks and can be easily integrated into existing software applications. KOCL is a powerful tool for developers who need to reduce the power consumption of their OpenCL applications in order to extend battery life and improve system performance.

KOCL is introduced in IEEE D&T 34(6)
Use Cases

• Hardware prototyping, design iteration
• Adaptive system deployment
  – Power-aware kernel selection
  – Fine-grained DVFS, clock gating, ...
• Fault, malware detection
• Billing
• ...


KAPow

- Hardware/software framework providing **power breakdowns** for arbitrary FPGA-based systems at **user-specified granularity**
KAPow

- Hardware/software framework providing **power breakdowns** for arbitrary FPGA-based systems at user-specified granularity

- Monitoring of switching activities
  - Power-indicative signals selected

- Online modelling
  - Compensates for changes in environment, workload

- System power measurements split by module
KAPow: Further Reading

Introduced at IEEE FCCM’16 (Best Paper)
KAPow: Further Reading

Introduced at IEEE FCCM’16 (Best Paper)

Extended in ACM TRETS 11(1)
Motivation

• Have existing fine-grained power estimation framework...
• ... but it requires HDL expertise
• “Hardware is hard” – can we hide it?
Motivation

• Have existing fine-grained power estimation framework...
• ... but it requires HDL expertise
• “Hardware is hard” – can we hide it?

• Aims:
  – Generality
  – Minimal user effort
  – Transparency
  – Low overheads
OpenCL for FPGAs

- Adopted as input language by Altera, Xilinx
- Front-ends to existing vendor tools
  - High-level synthesis
  - System integration
  - Mapping, placement, routing, ...
- Kernel code compiled offline...
  - 1 kernel = 1 hardware accelerator
- ... and stitched to supporting infrastructure
  - Global memory interfacing
  - Launching kernels
Developer Burden: Hardware
Developer Burden: Hardware

• Before:
  
  ```
  ./aoc <.cl file> --board <board name>
  ```

• After:
  
  ```
  ./koc <.cl file> --board <board name>
  ```
Developer Burden: Hardware

• Before:
  ```
  ./aoc <.cl file> --board <board name>
  ```

• After:
  ```
  ./koc <.cl file> --board <board name>
  ```

• Optional flags:
  ```
  - kernels Choose a subset of kernels to monitor
  - kapow_n Control fidelity of measurements
  - kapow_w
  ```
Developer Burden: Software

• Initialise:

```c
#include "KOCL.h"
KOCL_init(float <update period>);
```

– Controls reactivity of power model

• Use:

```c
KOCL_built();
KOCL_get(char* <kernel name>);
KOCL_get("static");
```

• Clean up:

```c
KOCL_del();
```
Vanilla Tool Flow

High-level synthesis, system integration

Kernel source (.cl) ➔ Generate system (aoc -s) ➔ Top-level QSys system

Interface QSys system
Kernel wrapper
Kernel Verilog

Compile (aoc) ➔ Bitstream

Mapping, placement, routing, ...
KOCL Tool Flow
KOCL Tool Flow: HDL

• Per kernel:
  – Compile $\rightarrow$ netlist
    • Specifies use of FPGA resources
  – Perform power simulation to obtain switching estimates
    • Fast
    • No user input
  – Augment $N$ most-switching signals with $W$-bit activity counters
  – Substitute for original HDL
KOCL Tool Flow: Interfacing 1

- Expose busses to allow counter control, readback
KOCL Tool Flow: Control

• Per kernel:
  – Add controller
  – Connect to counters in netlist
  – Parameterise with hash of kernel’s name
KOCL Tool Flow: Interfacing 2

- Connect controllers
KOCL Tool Flow: TTL

- Need to determine optimal measurement period
  - Too small: low dynamic range
  - Too large: potential overflow
- Read $f_{\text{max}}$ from compilation report
- Given $f_{\text{max}}, W$, calculate TTL
- Apply via controller ROMs
KOCL Software

• Launched by, runs alongside host code
• Python w/Numpy, C API
KOCL Software

- Launched by, runs alongside host code
- Python w/Numpy, C API

- Three threads:
  - Model
    - Talks to hardware
    - Performs power modelling
  - Interface
    - Responds to host code requests
  - Messenger
    - Model-interface communication
KOCL Software: Model

• Initialisation:
  – Establish kernel names from bitstream
  – Discover controllers in hardware
  – Match to kernel names using hashes
  – Read parameters \((N, W)\) from controllers
  – Construct model
KOCL Software: Model

- **Initialisation:**
  - Establish kernel names from bitstream
  - Discover controllers in hardware
  - Match to kernel names using hashes
  - Read parameters \((N, W)\) from controllers
  - Construct model

- **Every** `update_period`:
  - Get activity, system power measurements
  - Update model
  - Pass power breakdown to messenger
Results

• Things of interest:
  – **Accuracy**
    • Estimate vs measurement
  – Compilation time overhead
  – Area overhead
  – Power overhead
  – Max. model update rate
Results

• Things of interest:
  – **Accuracy**
    • Estimate vs measurement
  – Compilation time overhead
  – Area overhead
  – Power overhead
  – Max. model update rate

• Particularly dependent on choice of $N$
• Found $W = 9$ generally best accuracy-overhead compromise
Accuracy

**System 1**

**System 2**

- Converged error (mW)
- Counters per module $N$
Compilation Overheads

System 1

System 2

Counts per module $N$
Runtime Overheads

Power consumption

Model update time

Counters per module $N$

Overhead (%)
Further Work

• Improved signal selection
• Incorporation of macro modelling
• Use for system-level control
• More devices, vendors
• Similar tools for monitoring performance, reliability
Preliminary Improvements

Signal selection improved in FPL'17
Preliminary Improvements

Signal selection improved in FPL’17

Extension in the works...
Summary

• Framework providing **kernel-level power estimates of arbitrary OpenCL systems** executing on **Altera FPGAs to host code**

• Easy to use
  – No hardware exposure

• ⩾ order-of-magnitude accuracy improvement vs simulation

• Remains under active development
Summary

• Framework providing **kernel-level power estimates** of arbitrary **OpenCL systems** executing on **Altera FPGAs** to **host code**

• Easy to use
  – No hardware exposure

• ≥ order-of-magnitude accuracy improvement vs simulation

• Remains under active development

• Open source
  – [https://github.com/PRiME-project/KOCL](https://github.com/PRiME-project/KOCL)
  – Plug-and-play Linux image, demo apps included

• Please use and provide feedback!
Backup
KAPow: Monitoring

- Modules analysed to identify power-indicative signals
- Lightweight activity counters transparently inserted
KAPow: Modelling

- Activities + system power $\rightarrow$ module-level power
- Online training, refinement
  - Adapts to changes in voltage, temperature, workload, noise, ...

Diagram:
- Black box system
- Mathematical model $\hat{x}$
- Adaptive online algorithm (RLS)

Symbols:
- $\alpha$: activity counts
- $y$: measured power
- $\hat{y}$: estimated power
- $e$: error
- $\hat{x}$: model coefficients